Scheduling Independent Tasks with Voltage Overscaling

Aurélien Cavelan, Yves Robert, Hongyang Sun, Frédéric Vivien

Scheduling Independent Tasks with Voltage Overscaling

Aurélien Cavelan*, Yves Robert*†, Hongyang Sun*, Frédéric Vivien*
*Ecole Normale Supérieure de Lyon & INRIA, France
†University of Tennessee Knoxville, USA

***Abstract*—In this paper, we discuss several scheduling algorithms to execute independent tasks with voltage overscaling. Given a frequency to execute the tasks, operating at a voltage below threshold leads to significant energy savings but also induces timing errors. A verification mechanism must be enforced to detect these errors. Contrarily to fail-stop or silent errors, timing errors are deterministic (but unpredictable). For each task, the general strategy is to select a voltage for execution, to check the result, and to select a higher voltage for re-execution if a timing error has occurred, and so on until a correct result is obtained. Switching from one voltage to another incurs a given cost, so it might be efficient to try and execute several tasks at the current voltage before switching to another one. Determining the optimal solution turns out to be unexpectedly difficult. However, we provide the optimal algorithm for a single task, the optimal algorithm when there are only two voltages, and the optimal level algorithm for a set of independent tasks, where a level algorithm is defined as an algorithm that executes all remaining tasks when switching to a given voltage. Furthermore, we show that the optimal level algorithm is in fact globally optimal (among all possible algorithms) when voltage switching costs are linear. Finally, we report a comprehensive set of simulations to assess the potential gain of voltage overscaling algorithms.**

***Index Terms*—HPC, Resilience, Failures, Timing Errors, Voltage Overscaling, Energy Efficiency**

I. Introduction

Energy minimization has become a critical concern in High Performance Computing (HPC). Many authors have suggested to use *Dynamic Voltage and Frequency Scaling (DVFS)* to reduce the energy consumption during the execution of an application. Reducing the frequency (or speed) at which each core is operated is the most frequently advocated approach, and great savings have been demonstrated for a variety of scientific applications [7], [1], [4], [13]. However, reducing the voltage for a given speed may lead to even greater savings, because the total consumed power is proportional to the square of the voltage. On the contrary, the dynamic power is linearly proportional to the frequency, and the static power is independent of it.

Keeping the same frequency and reducing the voltage is a promising direction which we explore in this paper. There is no free lunch, though. Given a frequency, there is always a voltage recommended by the manufacturer, below which it might be unsafe to operate the core. This voltage always includes some environmental margin to be on the safe side. Near-threshold computing is a technique that consists in reducing the voltage below the recommended value, down to a *threshold voltage* V_{TH} (also called *nominal voltage*) that is still considered safe. Overscaling algorithms suggest to further reduce the operating voltage, at the risk of producing *timing errors*. Because the voltage is set to a very low value, the results of some logic gates could be used before their output signals reach their final values, which could possibly lead to an incorrect result. The occurrence of a timing error depends upon many parameters: the voltage and frequency, and the nature of the target operation: different operations within the ALU may have different critical-path lengths. But in addition, for a given operation, different sets of operands may lead to different critical-path lengths (to see this, take a simple addition and think of a carry rippling to different gates depending upon the operands). Timing errors are therefore very different from usual fail-stop failures or silent errors that are dealt with in the literature: they are not random but, instead, they are purely deterministic. Indeed, replaying the same operation with the same set of operand under the same conditions will lead to the same result. Although deterministic, timing errors are unpredictable, because it is not possible to test all possible operands for a given operation. Therefore, for a given operation[1], an error probability is associated with each voltage and represents the fraction of operands for which incorrect results will be produced by executing that operation on these operands at that voltage.

We need to take actions to mitigate timing errors that strike the application when voltage is aggressively lowered. After executing a task, we insert a verification mechanism to check the correctness of the result. In our study, the scheduling algorithms are agnostic of the nature of this verification mechanism, which could be anything from (costly) duplication to (cheap) checksumming and other application-specific methods. Of course, the cheaper the cost of the verification, the smaller the overhead.

To execute a given task, scheduling algorithms with voltage overscaling operate as follows: they are given a (discrete) set of possible voltages to operate with, and one of them as an input voltage. The first decision to take is whether to execute the task at that voltage or to choose another one. In the latter case, there is a *switching cost* to pay to change voltage. Regardless of the decision, the result is verified after the execution of the task. If the verification mechanism returns that the result is correct, we are done. If not, we need to re-execute the task. Remember that timing errors are deterministic: there is no point in re-executing the task with the same voltage; we know that we will get the same error. We need to select a higher voltage that will reduce the probability of failure, paying a switching cost, and re-execute the task with this voltage. Because the voltage is higher, the error probability is reduced, and we have a chance that the second execution is correct. The higher the second voltage, the better that chance, but the higher the cost of the execution, so there is

[1] and a given frequency: remember that throughout the text, we assume the frequency to be given.

a trade-off to consider. If we are unlucky, we may have to try several higher and higher voltages, until eventually finishing the task by using the threshold voltage, which is 100% safe but very costly. In Section IV, we give the optimal scheduling algorithm for a single task, extending our previous result [3] to the case where an input voltage is given to the algorithm.

The problem gets more complicated when there are many tasks to schedule. We assume that these tasks correspond to the same operation but involve different operands (think of a collection of matrix products or stencil updates). Given a voltage, each task has the same probability to fail. In the absence of switching costs (an unrealistic assumption in practice), the tasks can be dealt with independently. However, to amortize the switching cost from a given voltage to a new one, it might be a good idea to try and execute several tasks (or even all the remaining tasks) at a given voltage. One key contribution of this work is to analyze *level algorithms*, which always execute all the remaining tasks once a voltage has been selected. We provide a dynamic programming algorithm that computes the optimal level algorithm as a function of the voltage costs and error probabilities, and of the number of tasks to execute.

Level algorithms turn out to be dominant among all possible algorithms when voltage switching costs are linear. Technically, if we have three voltages $V_1 < V_2 < V_3$, linear switching costs means that $s_{1,3} = s_{1,2} + s_{2,3}$, where $s_{i,j}$ is the cost to switch from V_i to V_j (or the other way round, from V_j to V_i). With linear switching costs, we show that the optimal level algorithm is in fact optimal among all possible algorithms, not just level algorithms.

Finally, an important contribution of the paper is to experimentally assess the usefulness of voltage overscaling algorithms. We first consider a case study from numerical linear algebra, where tasks are matrix-products that can be verified through ABFT checksums. We then envision different scenarios, where we evaluate the impact of each parameter (verification cost, voltage cost and error probability, switching costs). In addition to the gain in energy consumption, we also investigate the performance degradation: while we keep the same frequency (thereby avoiding a global slowdown of the execution as is the case with DVFS), we do have two sources of performance overhead: (i) the verification mechanism, and (ii) the time lost due to re-execution(s) and voltage switching after timing errors.

To the best of our knowledge, this paper (together with our initial workshop paper [3]) presents the first algorithmic approach for voltage overscaling. Previous studies are hardware oriented and require special hardware mechanisms to detect timing errors [9], [11], [10], [5]. On the contrary, we propose scheduling algorithms that can be called by the operating system of the platform.

The rest of this paper is organized as follows. In Section II, we introduce a formal model for timing errors. In Section III, we illustrate several scheduling strategies by considering only two available voltages. We present the optimal algorithm for a single task in Section IV, and move to scheduling several tasks in Section V. We report the results of a comprehensive set of simulations to assess the impact and benefits of the voltage overscaling algorithms in Section VI. Finally, we provide concluding remarks in Section VII.

II. Model

We now present a formal model for timing errors. We introduce the main notations and assumptions, and investigate the impact of timing errors on the success and failure probabilities of tasks. Because timing errors are deterministic, conditional properties are completely different from what is usually enforced for the resilience of HPC applications.

A. Timing errors

As already mentioned, we focus on a fixed frequency environment (this frequency may have been chosen to achieve a given performance). Then timing errors depend upon the voltage selected for execution, and we model this with the following two assumptions:

Assumption 1. *Given an operation and an input I (the set of operands), there exists a* threshold voltage $V_{\text{TH}}(I)$*: using any voltage V below the threshold ($V < V_{\text{TH}}(I)$) will always lead to an incorrect result, while using any voltage above that threshold ($V \geq V_{\text{TH}}(I)$) will always lead to a successful execution. Note that different inputs for the same operation may have different threshold voltages.*

Assumption 2. *When an operation is executed under a given voltage V, there is a probability p_V that the computation will fail, i.e., produces at least one error, on a random input. This failure probability is computed as $p_V = |\mathcal{I}_f(V)|/|\mathcal{I}|$, where $\mathcal{I}$ denotes the set of all possible inputs and $\mathcal{I}_f(V) \subseteq \mathcal{I}$ denotes the set of inputs for which the operation will fail at voltage V. Equivalently, $\mathcal{I}_f(V)$ is the set of inputs whose threshold voltages are strictly larger than V, according to Assumption 1. For any two voltages V_1 and V_2 with $V_1 \geq V_2$, we have $\mathcal{I}_f(V_1) \subseteq \mathcal{I}_f(V_2)$ (Assumption 1), hence $p_{V_1} \leq p_{V_2}$.*

If a task consists of t identical operations, each executed at voltage V with error probability p_V, then the probability of successful execution of the task is $(1 - p_V)^t$: the larger the task, the greater the risk. Since timing errors are essentially silent errors, they do not manifest themselves until the corrupted data has led to an unusual application behavior, which may be detected long after the error has occurred, wasting the entire computation done so far. Hence, an error-detection mechanism (also called *verification* mechanism) is necessary to ensure timely detection of timing errors after the execution of each task. In this paper, we assume that this mechanism is given. All the algorithms presented in Sections IV and V are fully general and agnostic of the error-detection technique (checksum, error correcting code, coherence tests, etc.).

B. Notations

We consider a set $\mathcal{T} = \{T_1, T_2, \cdots, T_n\}$ of n tasks to be executed by the system. There is no precedence constraint among the tasks so that they are *independent*. All tasks share the same computational weight, including the work to verify the correctness of the result at the end. Hence, they have the same execution time and energy consumption under a fixed voltage. We apply *Dynamic Voltage Over-Scaling (DVOS)* to tradeoff between energy cost and failure probability. The platform can choose an operating voltage among a set $\mathcal{V} = \{V_1, V_2, \cdots, V_k\}$ of k discrete values, where $V_1 < V_2 < \cdots < V_k$. Each voltage V_ℓ has an *energy cost* per task c_ℓ that increases with the voltage, i.e.,

$c_1 < c_2 < \cdots < c_k$. Based on Assumption 2, each voltage V_ℓ also has a *failure probability* p_ℓ that decreases with the voltage, i.e., $p_1 > p_2 > \cdots > p_k$. We assume that the highest voltage V_k equals the *nominal* voltage V_{TH} with failure probability $p_k = 0$, thus guaranteeing error-free execution for all possible inputs. For convenience, we also use a *null* voltage V_0 with failure probability $p_0 = 1$ and null energy cost $c_0 = 0$. Here is a summary of these notations:

Voltages	V_0	V_1	V_2	$\cdots$	$V_k = V_{\text{TH}}$
Failure Prob.	$p_0 = 1$	p_1	p_2	$\ldots$	$p_k = 0$
Energy cost	$c_0 = 0$	c_1	c_2	$\ldots$	c_k

We further assume that DVOS is only applied to reduce energy consumption, and leaving the system at a voltage below V_{TH} after the execution of the task(s) is not allowed (because it would not be safe). This is formally stated in the following assumption.

Assumption 3. *The system is initially running at nominal voltage V_{TH} and its voltage must be reset back to V_{TH} when all computations are done.*

Switching the operating voltage also incurs an energy cost. Let $s_{\ell,h}$ denote the energy consumed to switch the system's operating voltage from V_ℓ to V_h. We have $s_{\ell,h} = 0$ if $\ell = h$ and $s_{\ell,h} > 0$ if $\ell \neq h$. Moreover, we make the following assumptions on the properties of the switching costs, which are true in many systems in practice.

Assumption 4. *The voltage switching costs of the system satisfy:*
- Symmetry: $s_{\ell,h} = s_{h,\ell}$ *for all* $V_\ell, V_h \in \mathcal{V}$;
- Dominance: $s_{\ell,h} \geq s_{\ell,p}$ *and* $s_{\ell,h} \geq s_{p,h}$ *for* $V_\ell \leq V_p \leq V_h$;
- Triangle inequality: $s_{\ell,h} \leq s_{\ell,p} + s_{p,h}$ *for* $V_\ell \leq V_p \leq V_h$.

Definition 1 (Linear switching costs)**.** *We have linear costs when the triangle inequality is always an equality: $s_{\ell,h} = s_{\ell,p} + s_{p,h}$ for all $V_\ell \leq V_p \leq V_h$.*

The objective is to minimize the expected total energy consumption by determining the optimal strategy to execute the set of tasks.

C. Success and failure probabilities

We now consider the implications of Assumptions 1 and 2 on the success and failure probabilities of executing a task following a sequence of voltages. For the ease of writing, we assume that the execution of each task has already failed under the null voltage V_0 (at energy cost $c_0 = 0$).

Lemma 1. *Consider a sequence $\langle V_1, V_2, \cdots, V_m \rangle$ of m voltages, where $V_1 < V_2 < \cdots < V_m$, under which a given task is executed. For any voltage V_ℓ, where $1 \leq \ell \leq m$, given that the execution of the task on a certain input has already failed under voltages $V_0, V_1, \cdots, V_{\ell-1}$, the probabilities that the task execution will fail and succeed respectively under voltage V_ℓ on the same input are given by*

$$\begin{aligned} \mathbb{P}(V_\ell\text{-fail} \mid V_0 V_1 \cdots V_{\ell-1}\text{-fail}) &= \frac{p_\ell}{p_{\ell-1}}, \\ \mathbb{P}(V_\ell\text{-succ} \mid V_0 V_1 \cdots V_{\ell-1}\text{-fail}) &= 1 - \frac{p_\ell}{p_{\ell-1}}. \end{aligned}$$

Proof. We prove the claimed probabilities using Assumptions 1 and 2. The task under study is the execution of some computation on some input I. Since this task execution has failed under voltages $V_0, V_1, \cdots, V_{\ell-1}$, we know that input I satisfies $I \in \bigcap_{h=0}^{\ell-1} \mathcal{I}_f(V_h) = \mathcal{I}_f(V_{\ell-1})$ (recall that $\mathcal{I}_f(V_h)$ is the set of inputs on which the computation fails under voltage V_h). Then, the task execution will fail again under voltage V_ℓ if the input satisfies $I \in \mathcal{I}_f(V_\ell) \subseteq \mathcal{I}_f(V_{\ell-1})$. Otherwise, the task execution will succeed. Given that the input is randomly chosen (we have no a priori knowledge on it), the failure probability is

$$\mathbb{P}(V_\ell\text{-fail} \mid V_0 V_1 \cdots V_{\ell-1}\text{-fail}) = \frac{|\mathcal{I}_f(V_\ell)|}{|\mathcal{I}_f(V_{\ell-1})|} = \frac{|\mathcal{I}_f(V_\ell)|/|\mathcal{I}|}{|\mathcal{I}_f(V_{\ell-1})|/|\mathcal{I}|} = \frac{p_\ell}{p_{\ell-1}},$$

and the success probability is

$$\mathbb{P}(V_\ell\text{-succ} \mid V_0 V_1 \cdots V_{\ell-1}\text{-fail}) = \frac{|\mathcal{I}_f(V_{\ell-1}) \setminus \mathcal{I}_f(V_\ell)|}{|\mathcal{I}_f(V_{\ell-1})|} = 1 - \frac{|\mathcal{I}_f(V_\ell)|}{|\mathcal{I}_f(V_{\ell-1})|} = 1 - \frac{p_\ell}{p_{\ell-1}}.$$

□

III. Case with Two Voltages

In this section, we focus on the special case where there are only two available voltages (i.e., $k = 2$). In the first study, for the sake of illustration, we assume that Assumption 3 does not hold and that the system is originally at voltage V_1. In the second study, we derive the optimal policy, for two voltages, assuming that Assumption 3 does hold (which will always be the case in the following sections).

A. Two voltages without Assumption 3

The system is initially at voltage V_1. We assume that there are only two tasks to execute. It is twice as expensive energy-wise to execute a task at V_2 than at V_1 ($c_1 = 1$ and $c_2 = 2$), the failure probability at V_1 is 80% ($p_1 = 0.8$), and switching voltage from V_1 to V_2 is as expensive as executing 20 tasks at V_1 ($s_{1,2} = 20$). If there were no switching costs ($s_{1,2} = 0$), no task would be executed at voltage V_1. Indeed, executing a task at voltage V_2 costs $c_2 = 2$, whereas the expected cost of executing a task at V_1 is $c_1 + p_1 \cdot c_2 = 1 + 0.8 \cdot 2 = 2.6$. If we could switch voltages for free, we would always switch to the nominal voltage to execute tasks. However, the switching cost is expensive here, and the system is initially at voltage V_1. We can envision three policies:

- Switch directly to V_2 to execute both tasks. This costs $s_{1,2} + 2 \cdot c_2 = 24$.
- Execute both tasks at V_1. If both executions succeed, which happens with probability $(1 - p_1)^2$, we are done. Otherwise, switch to V_2 and re-execute the failed task(s). The expected cost is $2 \cdot c_1 + (1 - (1 - p_1)^2) s_{1,2} + 2 \cdot p_1 \cdot c_2 = 24.4$.
- Execute the first task at V_1 and switch to V_2 if and only if its execution failed. The rationale is as follows. If the execution of the first task fails, whatever the result of the execution of the second task, we will have to switch to V_2 to re-execute the first task. Moreover, we have shown that executing a task at V_2 is cheaper than trying to execute it at V_1 and then re-executing it at V_2 in case of failure. Therefore, we save energy by not attempting to execute the second task at V_1. If the execution of the first task was

successful, we remain at V_1 in (the unlikely) case that the execution of the second case will also succeed and that we will never have to switch to V_2 (thus saving the huge switching cost). The expected cost of this policy is then $c_1+p_1(s_{1,2}+2\cdot c_2)+(1-p_1)(c_1+p_1(s_{1,2}+c_2)) = 23.92$.

The third policy is the optimal in this case. This example illustrates two important facts: (i) the optimal policy may be dynamic, that is, the decision at which voltage the next task should be executed depends on the success or failure of *other* tasks; (ii) switching costs can have a significant impact on the shape of the optimal solution.

B. Two voltages under Assumption 3

We now assume that we have n tasks to process, and that Assumption 3 holds (as it should). We have two cases to consider, depending on whether V_1 is used or not.

- V_1 is not used. The cost is then $n \cdot c_2$.
- V_1 is used. Then, whatever the policy, the two switching costs $s_{1,2}$ and $s_{2,1}$ will be payed. Executing a task at voltage V_2 costs c_2. Executing a task first at voltage V_1 costs: $c_1 + p_1 \cdot c_2$. Therefore, it may only be worth using V_1 if $c_1+p_1 \cdot c_2 \leq c_2 \Leftrightarrow c_1 \leq (1-p_1)c_2$. In this case, an optimal solution is to first execute *all* tasks at V_1, then the voltage must be switched back to V_2, and finally the failed tasks must be re-executed. Indeed, whatever happens, both switching costs are paid, and the solution is the same as if switching costs were zero. The expected cost is then $2s_{1,2} + n \cdot c_1 + n \cdot p_1 \cdot c_2$. This is better than the first policy if and only if:

$$n \cdot c_2 \geq 2s_{1,2}+n \cdot c_1+n \cdot p_1 \cdot c_2 \Leftrightarrow n \geq \left\lceil \frac{2s_{1,2}}{c_2(1-p_1)-c_1} \right\rceil .$$

The optimal strategy is thus: if $c_1 \leq (1-p_1)c_2$ and $n \geq \lceil \frac{2s_{1,2}}{c_2(1-p_1)-c_1} \rceil$, then switch to V_1, execute all tasks at V_1, switch to V_2 and re-execute all failed tasks; otherwise, stay at V_2 and execute all tasks at that voltage. Hence, the shape of the optimal strategy depends on the number of tasks to execute. Note that thanks to Assumption 3, the shape of the solution is simpler than in the previous study.

IV. Scheduling for a single task

We focus here on the special case where there is only one task to execute. We present an optimal algorithm for this case, extending our previous result [3] to the case where an input voltage is given to the algorithm. We need this extension to prepare for the general case with several tasks: indeed, consider a simple algorithm that proceeds task after task. For the first task, the input voltage is always V_{TH}, according to Assumption 3. But when the execution of the first task completes, the input voltage for the second task may well be different from V_{TH}, if for instance the algorithm tried, and succeeded with, a lower voltage.

We first define some notations. Let $E^*_{\text{ONE}}(V_h, V_\ell)$ denote the optimal expected energy needed to execute the task starting from voltage V_h, given that the task has previously failed under a sequence of lower voltages, the highest one among which is V_ℓ. Let $E^*_{\text{ONE}}(V_\ell)$ and $V^*_{\text{ONE}}(V_\ell)$ denote, respectively, the optimal expected energy and the optimal voltage needed to execute the task after it has just failed under voltage V_ℓ. The following theorem shows the optimal solution.

Theorem 1. *To execute a single task on a system with k voltages, the optimal expected energy consumption as well as the optimal sequence of voltages can be obtained by dynamic programming with complexity $O(k^2)$.*

Proof. Suppose the task has just failed under voltage V_ℓ and it is about to be executed at a higher voltage $V_h > V_\ell$. According to Lemma 1, the probability that the tasks will fail again under V_h is given by $\mathbb{P}(V_h\text{-fail} \mid V_\ell\text{-fail}) = p_h/p_\ell$. If the task is successfully completed at V_h, we can just reset the voltage from V_h back to the nominal voltage V_k. Otherwise, we need to determine the optimal voltage $V^*_{\text{ONE}}(V_h)$ to continue executing the task until successful completion. Thus, for any pair (V_h, V_ℓ) of voltages with $V_0 \leq V_\ell < V_h \leq V_k$, we can compute $E^*_{\text{ONE}}(V_h, V_\ell)$ by the following dynamic programming formulation:

$$E^*_{\text{ONE}}(V_h, V_\ell) = c_h + \left(1 - \frac{p_h}{p_\ell}\right) s_{h,k} + \frac{p_h}{p_\ell} \cdot \min_{h<p\leq k} \left\{ s_{h,p} + E^*_{\text{ONE}}(V_p, V_h) \right\} . \quad (1)$$

The table is initialized with $E^*_{\text{ONE}}(V_k, V_\ell) = c_k$ regardless of V_ℓ, and the entire table can be computed in $O(k^2)$ time.

The optimal expected energy to execute the task after it has just failed under voltage V_ℓ is therefore

$$E^*_{\text{ONE}}(V_\ell) = \min_{\ell<h\leq k} \left\{ s_{\ell,h} + E^*_{\text{ONE}}(V_h, V_\ell) \right\} , \quad (2)$$

and the optimal voltage to execute the task in this case is $V^*_{\text{ONE}}(V_\ell) = V_{h'}$, where $h' = \arg\min_{\ell<h\leq k} \{ s_{\ell,h} + E^*_{\text{ONE}}(V_h, V_\ell) \}$. Again, computing $E^*_{\text{ONE}}(V_\ell)$ and $V^*_{\text{ONE}}(V_\ell)$ for all $1 \leq \ell \leq k$ takes $O(k^2)$ time. The optimal expected energy to execute the task from the initial nominal voltage V_k is thus:

$$E^*_{\text{ONE}} = \min_{1\leq h\leq k} \left\{ s_{k,h} + E^*_{\text{ONE}}(V_h, V_0) \right\} ,$$

where V_0 is the null voltage with failure probability $p_0 = 1$. The optimal starting voltage to execute the task is $V^*_{\text{ONE}} = V_{h'}$, where $h' = \arg\min_{1\leq h\leq k} \{ s_{k,h} + E^*_{\text{ONE}}(V_h, V_0) \}$. This can be computed in $O(k)$ time. □

V. Scheduling for several tasks

We now consider the general case of executing a set of n independent tasks, where $n \geq 2$. All tasks correspond to the same computational operations, but may have different threshold voltages, because they operate on different data sets. The problem turns out to be more complicated than expected. We start, in Section V-A, with the introduction of two simple scheduling strategies, *task-by-task* and *level*, before sketching the description of a general scheduling algorithm. Then we show how to determine the optimal level algorithm in Section V-B. Finally, in Section V-C, we prove that the optimal level algorithm is in fact globally optimal among all possible scheduling algorithms when switching costs are linear.

A. Scheduling algorithms and strategies

We now provide an informal description of scheduling algorithms for independent tasks. There are two simple execution strategies, which we call *task-by-task* and *level*. The task-by-task strategy considers the tasks one after another, waiting for the successful completion of the current task before proceeding to the (first) execution of the next one. This

strategy relies upon the optimal algorithm for a single task with a given input voltage described in Section IV. After the successful execution of the current task, the platform voltage is set at some value V_h, which we use as the input voltage for applying the optimal single-task algorithm to the execution of the next task.

Although optimal for each task, this strategy may end up paying a high overall switching cost. For instance, if the optimal single-task algorithm always starts with some low voltage, say V_s, regardless of the input voltage V_h, then the task-by-task strategy will have to switch back down to V_s each time there is a timing error in the execution of the previous task.

To minimize the total switching cost, another strategy is to execute all tasks at a given voltage, before switching to another voltage and execute all remaining tasks at that other voltage, and so on. This *level* strategy goes voltage-by-voltage instead of task-by-task, executing all tasks at a given voltage before switching to another one (hence its name).

While very natural, the task-by-task and level strategies are not the only possible algorithms. In fact, a general scheduling algorithm proceeds as follows. At each step, we are given an input voltage (that of the last execution of a task, or V_{TH} initially) and the list of remaining tasks, together with their history (the last voltage tried for the execution of each task is recorded, with V_0 for initial condition). Then the algorithm selects one task in the list and one voltage V_{new} higher than the one recorded for this task[2], and executes the task at that voltage. A switching cost is paid if V_{new} is different from the input voltage. If the execution is successful, the task is removed from the list, and otherwise, the task stays in the list, and its history is updated to be V_{new}. The algorithm then proceeds to the next step, with V_{new} as input voltage. The key decision at each step of the scheduling algorithm is the selection of the new task and voltage pair, and this decision may well depend upon the number and history of the tasks in the list, the input voltage, and all the problem parameters (cost and error probability of each voltage, and switching costs). Altogether, we have a complex decision to make at each step, and it seems very difficult to prove the optimality of a scheduling algorithm in the general case.

B. Level algorithms

In this section, we formally define level algorithms, and we provide a dynamic programming algorithm to compute the optimal level algorithm.

Definition 2 (Level algorithms)**.** *A level algorithm executes a set of independent tasks as follows:*

1) *Select the initial voltage V;*
2) *Switch to voltage V and execute all remaining tasks;*
3) *Remove the successfully completed tasks from the set;*
4) *If there are still some tasks not successfully completed, choose the next, higher, voltage V to try and go to Step 2;*
5) *If the last voltage used is not V_k, switch to voltage V_k.*

We call the sequence of voltages tried by a level algorithm the voltage sequence*.*

[2]There would be no point in trying the recorded voltage again, or a lower one: we know that the execution of the task would fail again.

The level strategy guarantees that each voltage will be used at most once, so the voltage will change no more than k times during the execution of the entire task set. We have to determine the optimal sequence of voltages to characterize the optimal level algorithm. Before presenting the dynamic programming algorithm that solves this problem, we need a few notations.

Let $E^*_{\text{SET}}(i, V_h, V_\ell)$ denote the optimal expected energy needed to execute i tasks starting at voltage V_h, provided that these tasks have just failed under a lower voltage $V_\ell < V_h$. Then, the optimal expected energy to execute i tasks that have failed under voltage V_ℓ is given by $E^*_{\text{SET}}(i, V_\ell) = \min_{\ell<h\le k}\left\{s_{\ell,h} + E^*_{\text{SET}}(i, V_h, V_\ell)\right\}$. The following theorem shows the optimal solution of a level algorithm.

Theorem 2. *To execute a set of n independent tasks on a system with k voltages, the optimal solution under any level algorithm can be obtained by dynamic programming with complexity $O(n^2k^2)$.*

Proof. Suppose a set of i tasks have just failed under voltage V_ℓ and they will be executed at a higher voltage V_h. According to Lemma 1, the probability that any of these tasks will fail again at V_h is $\mathbb{P}(V_h\text{-fail} \mid V_\ell\text{-fail}) = p_h/p_\ell$. Thus, the probability that j tasks will remain uncompleted after they are executed at voltage V_ℓ is

$$\mathbb{P}(j \text{ tasks remain}) = \binom{i}{j}\left(\frac{p_h}{p_\ell}\right)^j\left(1-\frac{p_h}{p_\ell}\right)^{i-j},$$

for any $0 \le j \le i$. If no task remains, e.g., $j = 0$, we need to reset the voltage from V_h back to the nominal voltage V_k. Otherwise, we need to determine the optimal voltage to execute the remaining j tasks, hence the dynamic program:

$$\begin{aligned}
&E^*_{\text{SET}}(i, V_h, V_\ell) = i\cdot c_h + \mathbb{P}(\text{no task remains})\cdot s_{h,k}\\
&+\sum_{j=1}^{i}\left(\mathbb{P}(j \text{ tasks remain})\min_{h<p\le k}\left\{s_{h,p} + E^*_{\text{SET}}(j, V_p, V_h)\right\}\right)\\
&= i\cdot c_h + \left(1-\frac{p_h}{p_\ell}\right)^i s_{h,k}\\
&+\sum_{j=1}^{i}\left(\binom{i}{j}\left(\frac{p_h}{p_\ell}\right)^j\left(1-\frac{p_h}{p_\ell}\right)^{i-j}\cdot\min_{h<p\le k}\left\{s_{h,p}+E^*_{\text{SET}}(j, V_p, V_h)\right\}\right)
\end{aligned}$$

for all $1 \le i \le n$ and all (V_h, V_ℓ) pairs with $V_0 \le V_\ell < V_h \le V_k$. In particular, when $V_h = V_k$, we have $E^*(i, V_k, V_\ell) = i\cdot c_k$ for all $1 \le i \le n$ regardless of V_ℓ.

The optimal expected energy needed to execute i remaining tasks after they have just failed under voltage V_ℓ is therefore

$$E^*_{\text{SET}}(i, V_\ell) = \min_{\ell<h\le k}\left\{s_{\ell,h} + E^*_{\text{SET}}(i, V_h, V_\ell)\right\}. \qquad (3)$$

The optimal voltage to execute these tasks is $V^*_{\text{SET}}(i, V_\ell) = V_{h'}$, where $h' = \arg\min_{\ell<h\le k}\left\{s_{\ell,h} + E^*_{\text{SET}}(i, V_h, V_\ell)\right\}$. The optimal expected energy needed to execute all n tasks, given that the initial system voltage is the nominal voltage V_k, is

$$E^*_{\text{SET}} = \min_{1\le h\le k}\left\{s_{k,h} + E^*_{\text{SET}}(n, V_h, V_0)\right\},$$

where V_0 is the null voltage with failure probability $p_0 = 1$. The optimal starting voltage to execute the entire set is $V^*_{\text{SET}} = V_{h'}$, where $h' = \arg\min_{1\le h\le k}\left\{s_{k,h} + E^*_{\text{SET}}(n, V_h, V_0)\right\}$.

The complexity is clearly dominated by the computation of $E^*_{\text{SET}}(i, V_h, V_\ell)$ for all $1 \leq i \leq n$ and $V_0 \leq V_\ell < V_h \leq V_k$, which takes $O(n^2k^2)$ time. □

Theorem 2 shows that the optimal voltage to select after the current voltage is used depends on the number of remaining tasks. To demonstrate this point, consider an example with three voltages and 10 independent tasks. The energy costs of the voltages are $c_1 = 0.1$, $c_2 = 1$ and $c_3 = 5$, and the corresponding error probabilities are $p_1 = 0.8$, $p_2 = 0.5$ and $p_3 = 0$. The voltage switching costs are $s_{1,2} = s_{2,3} = 1$ and $s_{1,3} = 1.1$. According to Theorem 2, the optimal voltage to start executing the tasks can be shown to be V_1. Now, suppose V_1 has been used. We consider the following two cases.

- There is only 1 task left. In this case, switching first to V_2 and in case of failure then to V_3 incurs an expected cost of $s_{1,2}+c_2+\frac{p_2}{p_1}(s_{2,3}+c_3)+\left(1-\frac{p_2}{p_1}\right)s_{2,3} = 6.125$. On the other hand, switching directly to V_3 incurs a total cost of $s_{1,3}+c_3 = 6.1$. Hence, the better strategy in this case is to switch directly to V_3.
- There are 9 tasks left. Then, switching first to V_2 and then to V_3 incurs an expected cost of $s_{1,2} + 9c_2 + \left(1-\frac{p_2}{p_1}\right)^9 s_{2,3} + \sum_{j=1}^{9}\binom{9}{j}\left(\frac{p_2}{p_1}\right)^j\left(1-\frac{p_2}{p_1}\right)^{9-j}(s_{2,3} + j \cdot c_3) = 39.125$. On the other hand, switching directly to V_3 incurs a total cost of $s_{1,3}+9c_3 = 46.1$. Hence, the better strategy in this case is to try voltage V_2 first and then V_3.

Intuitively, using the intermediate voltage V_2 pays off only when there are many tasks left, in which case the extra switching overhead ($s_{1,2}+s_{2,3}-s_{1,3} = 0.9$) diminishes with respect to the potential energy gained from task execution.

C. Optimality result

In this section we prove that level algorithms are dominant when switching costs are linear. The analysis so far has assumed that the voltage switching cost follows triangle inequality. However, under the special case of linear costs, i.e., $s_{\ell,h} = s_{\ell,p} + s_{p,h}$, we can show that there exists a level algorithm (satisfying Definition 2) that is optimal. Furthermore, the optimal voltage sequence has a much simpler structure, which helps reduce significantly the complexity of the optimal algorithm.

We first prove a simple result: when the switching costs are zero, the optimal algorithm for a single task defines a level algorithm that is also optimal for an arbitrary number of tasks. Intuitively, we can transform any task-by-task algorithm into a level algorithm with the same cost, because there is no overhead to switch voltages.

Lemma 2. *On a system without voltage switching costs, there exists an optimal algorithm, which is a level algorithm, such that after each voltage:*

- *The optimal voltage to execute the remaining tasks does not depend on the number of remaining tasks, and it is the same as the optimal voltage to execute a single task;*
- *The optimal expected energy consumption is proportional to the number of remaining tasks.*

Proof. Let us consider an optimal algorithm $\mathcal{O}$ and its execution on an instance with n tasks denoted by T_1, ..., T_n. We reorder the task executions performed by $\mathcal{O}$ such that all executions of T_1 are done first, then all executions of T_2 are performed, and so on. (Note that we have taken an arbitrary ordering of the tasks.) The new execution order has exactly the same cost as the previous one because switching costs are null. Also, Assumption 3 has no impact on the solution. Therefore, the optimal sequence of voltages to execute task T_i (for any $1 \leq i \leq n$) is the sequence of voltages defined in Section IV, that is, for a single task. Let $V_{\pi(1)}$, ..., $V_{\pi(m)}$ be this sequence. Then algorithm $\mathcal{O}$, being optimal, tried for each task this sequence of voltages, in increasing voltages, until success. We finally reorder, once again at no cost, the task executions performed by algorithm $\mathcal{O}$: first all the executions at voltage $V_{\pi(1)}$, that is, the execution of all tasks at $V_{\pi(1)}$; then all executions at voltage $V_{\pi(2)}$, that is, the execution of all remaining tasks at voltage $V_{\pi(2)}$; and so on. We have, hence, defined a level algorithm following the voltage sequence defined for a single task, and whose expected energy cost is the same as that of the optimal algorithm $\mathcal{O}$. □

Theorem 3. *With linear switching costs, level algorithms are dominant.*

Proof. Let us consider any optimal algorithm $\mathcal{O}$ and an instance with n tasks. Let V_ℓ be the lowest voltage used by algorithm $\mathcal{O}$ during the entire execution. Then, the total voltage switching cost incurred by $\mathcal{O}$ is $S^{\mathcal{O}} \geq s_{k,\ell} + s_{\ell,k}$. (Note that we use $s_{k,\ell}+s_{\ell,k}$ for clarity here, to emphasize that we switch down to V_ℓ and back to $V_k = V_{\text{TH}}$, but remember that $s_{k,\ell} = s_{\ell,k}$ by Assumption 4.)

Let us consider any level algorithm $\mathcal{L}$ that starts by switching to voltage V_ℓ. Then, whatever the voltages it uses among the voltages V_ℓ, $V_{\ell+1}$, ..., V_k, it incurs a total switching cost exactly equal to $S^{\mathcal{L}} = s_{k,\ell}+s_{\ell,k}$, because switching costs are linear. Therefore, we can assume without loss of generality that algorithm $\mathcal{L}$ switches to all the voltages V_ℓ, $V_{\ell+1}$, ..., V_k (maybe without executing any task at some of those levels). Then, the optimization problem of determining at which next voltage should the remaining tasks be executed is exactly the same as if there were no switching costs (there is no penalty incurred when an intermediate voltage is used). Then, Lemma 2 tells us not only what is the optimal level algorithm in such a case but, also, that it is optimal among all existing algorithms, hence the optimality of $\mathcal{L}$ among all algorithms using voltages among V_ℓ, $V_{\ell+1}$, ..., V_k. Because $\mathcal{O}$ is one of these algorithms and is optimal, we can conclude. □

Theorem 4. *To execute a set of n independent tasks on a system with k voltages and linear switching costs, the optimal solution can be obtained with complexity $O(k^2)$.*

Proof. According to Theorem 3, we only need to focus on level algorithms to obtain the optimal solution.

Suppose that a level algorithm starts executing the tasks at voltage V_h. Then, the total switching cost paid by the algorithm during the entire execution is given by $s_{k,h}+s_{h,k}$, which is fixed and does not depend on the sequence of voltages used. It remains to find the optimal expected energy consumption to execute the tasks from V_h without considering the voltage switching costs. When there are no voltage switching costs, let $\bar{E}^*_{\text{SET}}(i, V_h, V_\ell)$ and $\bar{E}_{\text{ONE}}(V_h, V_\ell)$ denote, respectively, the optimal expected energy to execute i tasks and one task

by starting from voltage V_h, given that all of them have previously failed under voltage V_ℓ. According to Lemma 2, we have $\bar{E}^*_{\text{SET}}(n, V_h, V_0) = n \cdot \bar{E}^*_{\text{ONE}}(V_h, V_0)$. We can then try all possible starting voltages to get the optimal expected total energy consumption as follows:

$$E^*_{\text{SET}} = \min_{1 \le h \le k} \left\{ s_{k,h} + s_{h,k} + n \cdot \bar{E}^*_{\text{ONE}}(V_h, V_0) \right\} ,$$

and the optimal starting voltage is therefore $V_{h'}$, where $h' = \arg\min_{1 \le h \le k} \left\{ s_{k,h} + s_{h,k} + n \cdot \bar{E}^*_{\text{ONE}}(V_h, V_0) \right\}$.

Lemma 2 also shows that the optimal sequence of voltages to follow is the same as the optimal sequence to execute one task without considering voltage switching costs. According to Theorem 1, this can be computed by

$$\bar{E}^*_{\text{ONE}}(V_\ell) = \min_{\ell < h \le k} \bar{E}^*_{\text{ONE}}(V_h, V_\ell) ,$$

and $\bar{V}^*_{\text{ONE}}(V_\ell) = V_{h'}$ with $h' = \arg\min_{\ell < h \le k} \bar{E}^*_{\text{ONE}}(V_h, V_\ell)$. The complexity is determined by the computation of $\bar{E}^*_{\text{ONE}}(V_h, V_\ell)$ for all $0 \le V_\ell < V_h \le V_k$, which takes $O(k^2)$ time. □

In the general case, we may have triangle inequality but not triangle equality. We have not been able to design a counter-example to the optimality of level algorithms in the general case. We therefore conjecture the dominance of level algorithms even in the case of triangle inequality.

VI. Simulations

In this section, we evaluate the performance of the proposed algorithms using simulations. We instantiate the performance model with two scenarios. The first scenario is based on the available data about timing error probabilities on a specific hardware [5], and we consider a set of matrix products using ABFT as the verification mechanism. In the second scenario, we use synthetic data to assess the impact of different parameters (verification cost, error probability, switching cost) on alternative hardware and applications.

A. Comparing algorithms

We compare our dynamic programming algorithm designed for a set of independent tasks, which we denote by *DP-indep*, to the following algorithms in the evaluation.

- *Baseline* & *Threshold*: These two are static algorithms that use the *environmental margin* voltage and *nominal* voltage, respectively, to execute the tasks. The former does not make use of any voltage scaling technique, and the latter scales the voltage down to achieve near-threshold computing. Both algorithms do not incur timing errors and hence do not require verification and re-execution.
- *DP-single*: This algorithm uses the dynamic programming solution for a single task and applies the optimal sequence of voltages to execute all tasks in the set one after another, using the output voltage of the last task as the input voltage for the current task.

B. Case study: Matrix multiplication on FPGA

In the first evaluation scenario, we consider a concrete application (matrix multiplication) executed on a specific platform, where error probabilities are known from real measurements.

1) Platform setting: We adopt the set of voltages and error probabilities measured by Ernst et al. [5] on an FPGA multiplier block. Figure 1 shows the error probability $p^{(1)}_\ell$ of each available voltage V_ℓ when performing a *single* operation with random inputs. We take the *zero margin* 1.54V as the nominal voltage and consider the *environmental margin* 1.69V as the base operating voltage. As in [3], we scale the error probabilities down by a factor of 10 to account for the circuit-level error recovery technique [5]. Since the dynamic power consumption is a quadratic function of the operating voltage [2], [12], for a given voltage V_ℓ, the energy consumed to execute a task (one matrix product) is modeled as $c_\ell = V_\ell^2 w$, where w denotes the total number of operations in the task. The energy to switch the operating voltage is assumed to be linear (thus following triangle equality), and it is modeled as $s_{\ell,h} = \beta \cdot \frac{|V_\ell - V_h|}{V_k - V_1}$, where β captures the relative cost of voltage switching in comparison to computation.

2) Application modeling: We consider the computation of a set of matrix products of the same size, which forms a set of independent tasks of the same computational cost. Each product consist of m^3 multiply-add operations, where m denotes the size of the matrices. To detect errors we employ Algorithm-Based Fault Tolerance (ABFT) [8], which uses checksums to detect, locate and even correct errors in many linear algebra kernels. Specifically, by adding one (column or row) checksum to each of the input matrices, the technique enables to detect and correct up to one error during the computation of a matrix product with an overhead of $O(m^2)$ additional operations. This is almost negligible compared to the $O(m^3)$ operations incurred by the raw computation for reasonable matrix sizes. In the simulation, we fix the matrix size to be $m = 64$, so the ABFT version has $w = m(m+1)^2 = 64 \times 65^2$ operations, incurring an overhead of about 3%. With the ability to correct one error, the probability of having an incorrect product using voltage V_ℓ is thus given by $p_\ell = 1 - \left(1 - p^{(1)}_\ell\right)^w - \binom{w}{1} \left(1 - p^{(1)}_\ell\right)^{w-1} p^{(1)}_\ell$.

3) Results: Figure 2 presents the impact of the number of tasks n on the expected energy consumption when the voltage switching cost β is set to be equivalent to multiplying two matrices of size 64×64, which is almost the cost of executing one task. When the number of tasks is small (e.g., less than 7), the switching cost can not be amortized and the best choice is to stay at nominal voltage. Additionally, the cost of the verification mechanism (ABFT) when $m = 64$ reaches 3% of the total work and both *DP-single* and *DP-indep* remain worse than executing all tasks at nominal voltage and without any verification mechanism. However, when the number of tasks is large enough (e.g., more than 7), *DP-indep* quickly outperforms *DP-single*, which only focuses on one task at a time and is thus unable to lower the voltage if it is not worth it for one task. On the contrary, *DP-indep* is paying the switching cost only once, which can be easily amortized over the execution of the entire set of tasks. In particular, when the number of tasks reaches $n = 32$, *DP-indep* provides 5% of energy savings compared to the *Threshold* algorithm, and it saves more than 20% of energy compared to the *Baseline* algorithm.

Figure 3 shows the impact of the switching cost β on the expected energy consumption of *DP-single* and *DP-indep*

under different switching costs when the number of tasks is fixed to 32. Again, we model the switching cost β to be equivalent to multiplying two matrices of size $x \times x$. The x axis shows the corresponding matrix size. When the switching cost is small (e.g., $x = 20$), both *DP-single* and *DP-indep* yield the same expected energy consumption. In fact, they are both able to amortize the switching cost with one task and they use the same sequence of voltages. But as the switching cost increases, *DP-single* quickly shows its limits while *DP-indep* manages to better amortize the overhead and remains better than *Threshold* even when the switching cost is more than three times the cost of one task (e.g., $x = 96$). Note that when the switching cost is high enough (e.g., $x > 116$), both algorithms are unable to perform better than *Threshold*.

Overall, when the number of tasks is large enough, or if the switching cost is small, *DP-indep* is always better and it saves up to 23% of the expected energy compared to *Baseline* and up to 7% compared to *Threshold* in this configuration.

C. Evaluations with synthetic data

We now consider synthetic input data in order to evaluate the algorithms on different hardware and applications. We envision platforms with normalized voltages falling in [0.5, 1], where 1 represents the nominal voltage V_k. Following the characteristics of soft errors [6], we model the error probability of a computation of length w executed using voltage V_ℓ to be $p_\ell = 1 - e^{-\lambda_\ell w}$, where $\lambda_\ell = \lambda_0 e^{-c(V_\ell - V_k)} - \lambda_0$ represents the error rate and c denotes the failure rate coefficient that depends on the hardware. The model also specifies the base error rate λ_0, which is usually very small. In the experiments, λ_0 is fixed to be 10^{-5} error per second, which corresponds to less than one error per day. The energy consumption to execute the tasks and the voltage switching costs follow the same model as in the previous scenario.

1) Results: For all experiments, the total work is fixed to be $W = 10000$ operations and the number of tasks is fixed at $n = 32$, so that each task has about $w = \frac{W}{n} \approx 312$ operations.

Figure 4(a) shows, given a voltage, the probabilities of failure for one task under different failure rate coefficients c. This factor determines how *fast* the probability of failure increases when the voltage is lowered below threshold. A small value of c gives a very optimistic configuration, where lowering the voltage below threshold is possible while keeping low probabilities of failure. Such a configuration is ideal and allows us to choose amongst a wide range of voltages, thus giving the opportunity to save energy. Higher values of c show more realistic, and also more pessimistic configurations. For example, when c reaches 128, as soon as we lower the voltage, the probability of failure increases dramatically and the chance of success drops close to zero. In that case, *DP-indep* has no other choice but to stay at nominal voltage.

Figure 4(b) presents the impact of the number of tasks on the normalized expected energy consumption of *DP-indep* with respect to *Threshold* under different values of c. Remember that the total work W is fixed to be 10000, so when the number of task decreases, the size of the tasks increases and the probability of success to execute one task drops considerably. As a result, we have to use higher voltages, which consumes more energy.

The value of c determines the range of voltages that can be used safely, i.e., with low probability of failure, as shown by Figure 4(a) and it has a huge impact on the expected energy consumption. With a small c, the algorithm can yield important energy savings compared to *Threshold* (more than 40% for 10 tasks with $c = 4$), while with a large c (e.g., $c = 128$) the algorithm will not be able to do better than *Threshold*.

Finally, Figure 4(c) evaluates the impact of the verification cost on the expected energy consumption of *DP-indep*. In this experiment, we vary the verification cost with respect to the cost of one task from 0% (no verification cost) to 100% (the verification cost is equivalent to the cost of one task). When the cost of the verification increases, so does the total overhead of the computation. Consider the case where the cost of the verification is half the cost of one task (i.e., total work including verifications is now $1.5W = 15000$). In this case, executing 256 tasks is 40% more energy-efficient than *Threshold*, whereas under the same configuration, executing only 32 tasks for the same amount of work is only 5% more energy-efficient. Overall, the execution of many small tasks is preferred over the execution of a few big tasks.

2) Optimal solution with two voltages: We now consider a set of scenarios where only two voltages are available, namely, V_1 and $V_k = V_{\text{TH}}$. As shown previously, the only decision to make is whether to switch to the lower voltage V_1, in which case a cost of $2s_{k,1}$ is incurred due to voltage switching, or to directly use the nominal voltage V_k for executing the tasks. Figure 5 shows the energy savings achieved by the optimal algorithm (*DP-indep*) for this scenario. Naturally, more energy is saved when V_1 has a smaller execution cost or a lower error probability. For any given cost and probability, the saving also increases with the number of tasks, because the cost of voltage switching can be better amortized. For the case with 10 tasks and $s_{k,1} = 5c_1$, and when the error probability of voltage V_1 is around 0.5, *DP-indep* starts to save energy when the cost c_1 drops below $c_k/3$, and it is able to save at least 40% of the energy with a cost lower than $c_k/10$.

For the same case (10 tasks and $s_{k,1} = 5c_1$), Figure 6 shows the performance degradation (expected execution time overhead) due to verifications and re-executions. Since the results are obtained by the *DP-indep* algorithm, which targets energy minimization, they do not represent the best possible performance. Indeed, the overhead is minimum (0%) when the algorithm does not switch to V_1, which means that no energy is saved at all. Once the algorithm has decided to make the switch in order to save energy, the expected performance starts to degrade. For any fixed cost ratio c_k/c_1, a lower error probability enables the algorithm to both save energy and improve performance, because the tasks enjoy a higher chance of success at V_1. Finally, as verification and/or voltage switching times increase, the performance degrades further. For instance, when $p_1 = 0.5$ and $c_k/c_1 = 10$, and when verification takes 5% of the task execution time and voltage switching takes the same time as task execution, the algorithm achieves 40% of the energy saving at the expense of about 70% degradation in performance.

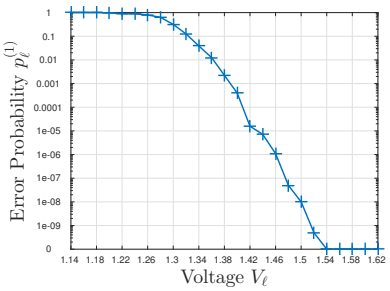


Figure 1: Error probabilities of available voltages measured on an FPGA multiplier block for a single operation with random inputs [5].

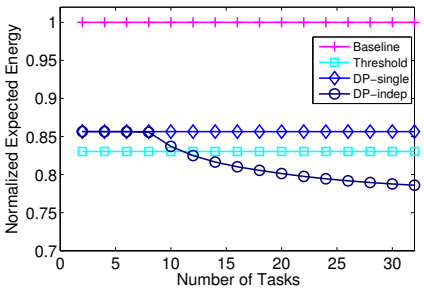


Figure 2: Impact of the number of tasks n on the energy consumption of the algorithms.

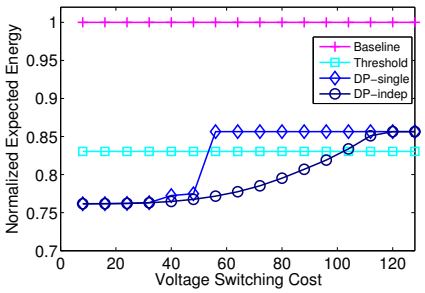
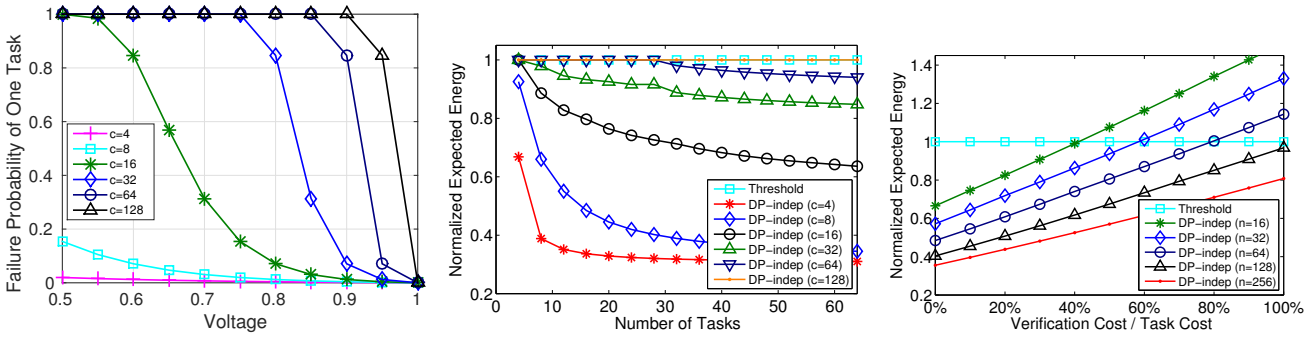


Figure 3: Impact of the voltage switching cost β on the energy consumption of the algorithms.



(a) (b) (c)

Figure 4: Impact of failure rate coefficient c on the failure probability of one task (a), and on the expected energy consumption (b). Impact of the verification cost on the expected energy consumption (c).

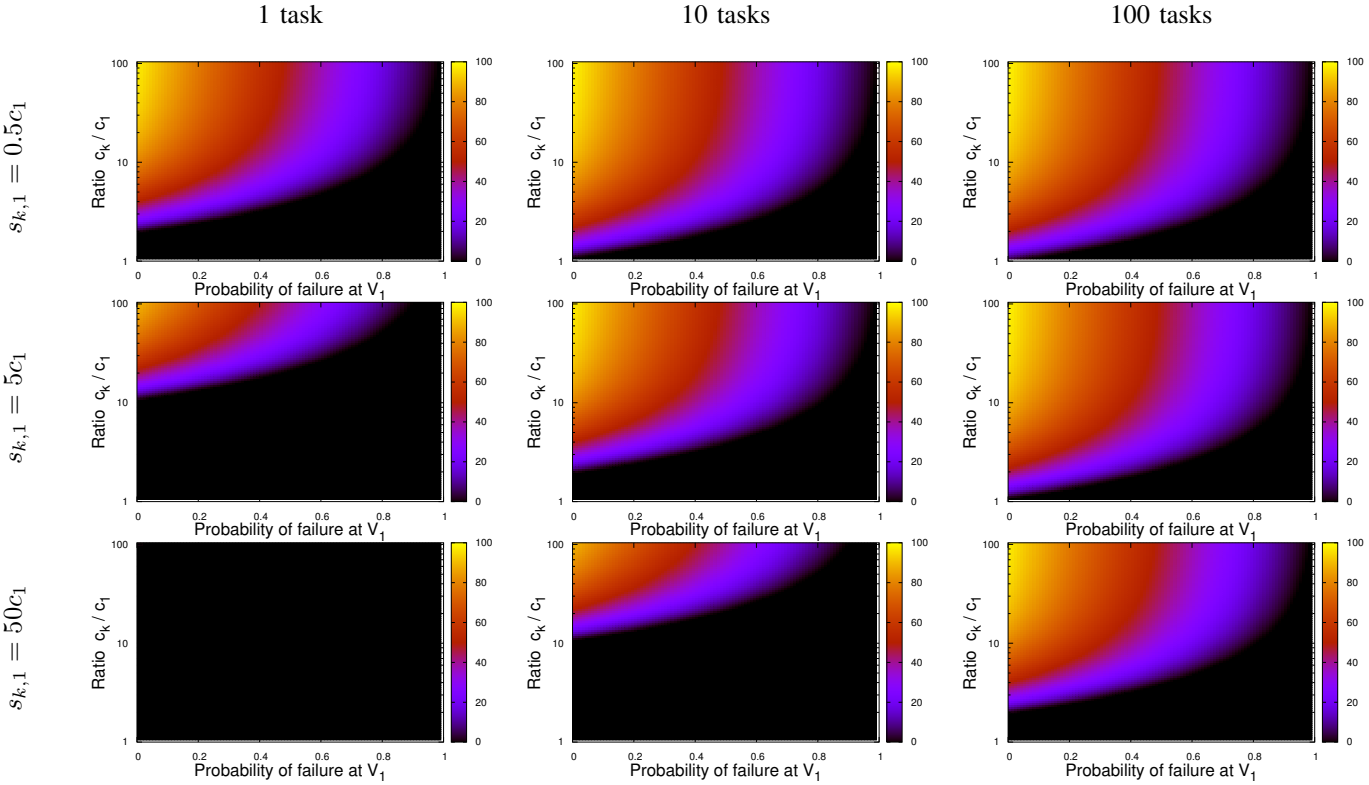


Figure 5: Impact of the switching cost and of the number of tasks on the percentage of energy saving when only two voltages are available. Black means no saving, and yellow means 100% of energy is saved.

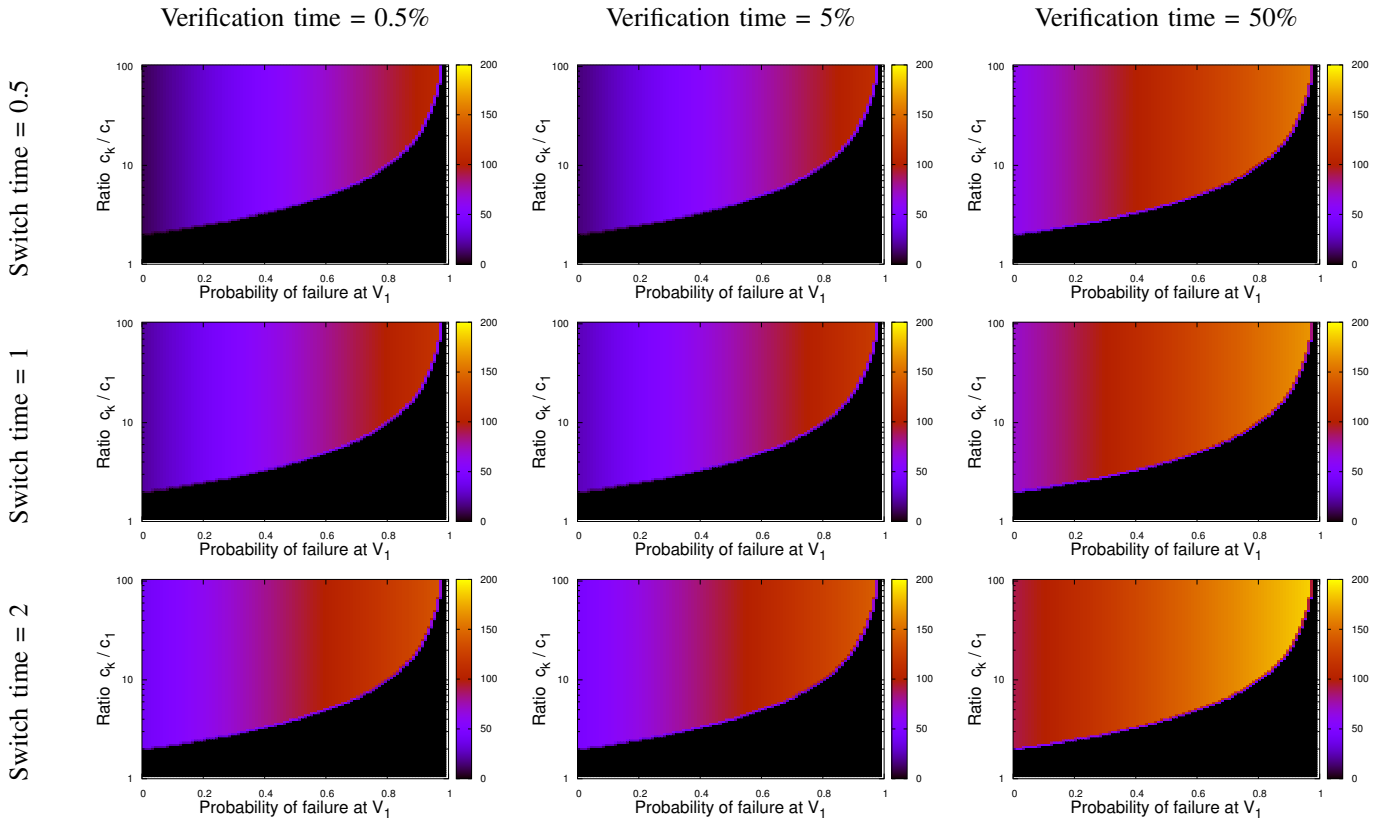


Figure 6: Percentage of (expected) execution time overhead as a function of the probability of failure and of the ratio of energy costs. The results are based on 10 tasks and a switching cost of $5c_1$. The unit used to express verification and switching times is the execution time of a task at voltage V_k.

VII. Conclusion

In this paper, we have used voltage overscaling to design a purely software-based approach for reducing the energy consumption of HPC applications. This approach aggressively lowers the supply voltage below the nominal voltage, introducing timing errors. Based on a formal model of timing errors, we have provided an optimal level algorithm to schedule a set of independent tasks, and we have proven its global optimality when switching costs are linear. The evaluation results obtained both for matrix multiplication on FPGA and for synthetic data demonstrate that our approach indeed leads to significant energy savings compared to the standard algorithm that always operates at (or above) the nominal voltage. Given the promising results, we plan to extend this approach to more complex application workflows in the future.

Acknowledgments. This research was funded in part by the European project SCoRPIO, by the LABEX MILYON (ANR-10-LABX-0070) of Université de Lyon, within the program "Investissements d'Avenir" (ANR-11-IDEX-0007) operated by the French National Research Agency (ANR), by the PIA ELCI project, and by the ANR RESCUE project. Yves Robert is with Institut Universitaire de France.

References

[1] N. Bansal, T. Kimbrel, and K. Pruhs. Speed scaling to manage energy and temperature. *Journal of the ACM*, 54(1):1 –39, 2007.

[2] D. M. Brooks, P. Bose, S. E. Schuster, H. Jacobson, P. N. Kudva, A. Buyuktosunoglu, J.-D. Wellman, V. Zyuban, M. Gupta, and P. W. Cook. Power-aware microarchitecture: Design and modeling challenges for next-generation microprocessors. *IEEE Micro*, 20(6):26–44, 2000.

[3] A. Cavelan, Y. Robert, H. Sun, and F. Vivien. Voltage overscaling algorithms for energy-efficient workflow computations with timing errors. In *Proceedings of 5th FTXS Workshop*, 2015.

[4] J.-J. Chen and C.-F. Kuo. Energy-efficient scheduling for real-time systems on dynamic voltage scaling (DVS) platforms. In *Proc. Int. Works. on Real-Time Computing Systems and Applications*, 2007.

[5] D. Ernst, S. Das, S. Lee, D. Blaauw, T. Austin, T. Mudge, N. S. Kim, and K. Flautner. Razor: circuit-level correction of timing errors for low-power operation. *IEEE Micro*, 24(6):10–20, 2004.

[6] F. Firouzi, M. E. Salehi, F. Wang, and S. M. Fakhraie. An accurate model for soft error rate estimation considering dynamic voltage and frequency scaling effects. *Microelectronics Reliability*, 51(2), 2011.

[7] R. Ge, X. Feng, and K. W. Cameron. Performance-constrained distributed DVS scheduling for scientific applications on power-aware clusters. In *Proceedings of SC'05*, page 34, 2005.

[8] K.-H. Huang and J. A. Abraham. Algorithm-based fault tolerance for matrix operations. *IEEE Trans. Comput.*, 33(6):518–528, 1984.

[9] G. Karakonstantis and K. Roy. Voltage over-scaling: A cross-layer design perspective for energy efficient systems. In *European Conference on Circuit Theory and Design (ECCTD)*, pages 548–551, 2011.

[10] P. Krause and I. Polian. Adaptive voltage over-scaling for resilient applications. In *Proceedings of DATE*, pages 1–6, 2011.

[11] S. Ramasubramanian, S. Venkataramani, A. Parandhaman, and A. Raghunathan. Relax-and-retime: A methodology for energy-efficient recovery based design. In *Proceedings of DAC*, 2013.

[12] N. B. Rizvandi, A. Y. Zomaya, Y. C. Lee, A. J. Boloori, and J. Taheri. Multiple frequency selection in DVFS-enabled processors to minimize energy consumption. In A. Y. Zomaya and Y. C. Lee, editors, *Energy-Efficient Distributed Computing Systems*. John Wiley & Sons, Inc., 2012.

[13] L. Wang, G. von Laszewski, J. Dayal, and F. Wang. Towards energy aware scheduling for precedence constrained parallel tasks in a cluster with DVFS. In *Proceedings of IEEE/ACM CCGRID*, 2010.